Computation of winning strategies for $\mu$-Calculus by fixpoint iteration

Christian Neukirchen

TCS Oberseminar · July 11, 2014
Overview

- Short introduction to $\mu$-calculus
- Parity games and strategies
- Strategies for $\mu$-calculus
- Example: mutex
- Implementation and optimization
- Future
Labelled Transition Systems

We consider LTS having a non-empty set of states $S$, total relations $a \in S \times S$ (for actions $a \in \mathcal{A}$) and propositions $p \in \mathcal{P}$ which hold at a state or not.
\( \mu \)-calculus: Syntax (from Hofmann and Rueß 2014)

\[
\phi ::= X \\
| \quad p \quad \neg p \\
| \quad [a]\phi \quad \text{(for all } a\text{-transitions)} \\
| \quad \langle a \rangle \phi \quad \text{(} a\text{-transition exists)} \\
| \quad \phi_1 \land \phi_2 \quad \phi_1 \lor \phi_2 \\
| \quad \mu X.\phi \quad \text{(least fixpoint)} \\
| \quad \nu X.\phi \quad \text{(greatest fixpoint)}
\]

Only propositions can be negated to ensure monotonicity.
μ-calculus: Examples

“q holds everywhere along the path”

$$\nu X. q \land [a] X$$

“q holds infinitely often on the path”

$$\nu X. \mu Y. (q \land \langle a \rangle X) \lor \langle a \rangle Y$$

“p holds at every even position” (more powerful than CTL*)

$$\nu X. p \land \langle a \rangle \langle a \rangle X$$
**μ-calculus: Set semantics**

\[
\begin{align*}
\text{sem}(X, \eta) &= \eta(X) \\
\text{sem}(\phi_1 \land \phi_2, \eta) &= \text{sem}(\phi_1, \eta) \cap \text{sem}(\phi_2, \eta) \\
\text{sem}(\phi_1 \lor \phi_2, \eta) &= \text{sem}(\phi_1, \eta) \cup \text{sem}(\phi_2, \eta) \\
\text{sem}(\lbrack a \rbrack \phi, \eta) &= \widetilde{pre}(\rightarrow)(\text{sem}(\phi, \eta)) \\
\text{sem}(\langle a \rangle \phi, \eta) &= \text{pre}(\rightarrow)(\text{sem}(\phi, \eta)) \\
\text{sem}(\mu X. \phi, \eta) &= \text{iter}_X(\phi, \eta, \emptyset) \\
\text{sem}(\nu X. \phi, \eta) &= \text{iter}_X(\phi, \eta, S) \\
s \in \widetilde{pre}(\rightarrow)(U) &\iff \forall t \in S. s \xrightarrow{a} t \implies t \in U \\
s \in \text{pre}(\rightarrow)(U) &\iff \exists t \in S. s \xrightarrow{a} t \land t \in U \\
\text{iter}_X(\phi, \eta, U) &= \text{let } U' := \text{sem}(\phi, \eta[X := U]) \text{ in} \\
&\quad \text{if } U = U' \text{ then } U \text{ else } \text{iter}_X(\phi, \eta, U')
\end{align*}
\]
Parity games

A parity game consists of a disjoint sum of positions $\text{Pos} = \text{Pos}_0 \cup \text{Pos}_1$, a total edge relation $\rightarrow \subseteq \text{Pos} \times \text{Pos}$ and a priority function $\Omega : \text{Pos} \rightarrow \mathbb{N}$.

Moves happen along the edge relation. The destination decides who moves next.

The game is won if the largest priority that occurs infinitely often is even, the opponent wins if it is odd.
Strategies for parity games

A strategy $\rho$ is a function that tells the player how to move next.

A positional strategy only takes the current position into account.

A position is in a winning set $W_i$ if there exists a strategy $\rho$ such that player $i$ wins, starting at a position in $W_i$.

**Theorem 1.** Every position $p$ is either in $W_0$ or $W_1$ and player $i$ wins positionally from every position in $W_i$. 
Strategies for $\mu$-calculus

We can interpret a $\mu$-calculus formula $\phi$ as a parity game. Moves can happen along the subformulae (example next slide). The priority of a position depends on the kind of formula and its nesting depth.

A partial winning strategy for $\mu$-calculus is a partial function

$$\Sigma : \Phi \times S \rightarrow s \quad (\text{move to state } s \in S)$$

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>*</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>(take the left formula)</td>
<td>(take the right formula)</td>
<td>(take the only formula)</td>
</tr>
</tbody>
</table>
Small strategy example

\[ \nu X. \mu Y. \left( q \land \langle a \rangle X \right) \lor \langle a \rangle Y \]

\[
X := \nu X. Y \\
Y := \mu Y. \left( q \land \langle a \rangle X \right) \lor \langle a \rangle Y
\]

\[
\begin{align*}
X & \quad 0 \rightarrow \ast \\
X & \quad 1 \rightarrow \ast \\
Y & \quad 0 \rightarrow \ast \\
Y & \quad 1 \rightarrow \ast \\
q & \quad 1 \rightarrow \ast \\
\langle a \rangle X & \quad 0 \rightarrow X & 1 \\
\langle a \rangle X & \quad 1 \rightarrow X & 1 \\
\langle a \rangle Y & \quad 0 \rightarrow Y & 1 \\
\langle a \rangle Y & \quad 1 \rightarrow Y & 1 \\
q \land (\langle a \rangle X) & \quad 1 \rightarrow \ast \\
(q \land (\langle a \rangle X)) \lor (\langle a \rangle Y) & \quad 0 \rightarrow \#2 \\
(q \land (\langle a \rangle X)) \lor (\langle a \rangle Y) & \quad 1 \rightarrow \#1
\end{align*}
\]
Updating strategies

Given two winning strategies \( \Sigma \) and \( \Sigma' \), we can define the partial winning strategy \( \Sigma + \Sigma' \) as

\[
(\Sigma + \Sigma')(\phi, s) = \begin{cases} 
\Sigma(\phi, s) & \text{if } (\phi, s) \in \text{dom}(\Sigma) \\
\Sigma'(\phi, s) & \text{else}
\end{cases}
\]
Strategy semantics

\[ \text{SEM}(X)_\eta = \{ (X, s) \mapsto * \mid s \in \eta(X) \} \]

\[ \text{SEM}(p)_\eta = \{ (p, s) \mapsto * \mid p \text{ holds at } s \} \]

\[ \text{SEM}(\neg p)_\eta = \{ (p, s) \mapsto * \mid p \text{ does not hold at } s \} \]

\[ \text{SEM}(\phi \land \psi)_\eta = \text{SEM}(\phi)_\eta + \text{SEM}(\psi)_\eta \]
\[ + \{ (\phi \land \psi, s) \mapsto * \mid (\phi, s) \in \text{dom}(\text{SEM}(\phi)_\eta) \]
\[ \land (\psi, s) \in \text{dom}(\text{SEM}(\psi)_\eta) \} \]

\[ \text{SEM}(\phi \lor \psi)_\eta = \text{SEM}(\phi)_\eta + \text{SEM}(\psi)_\eta \]
\[ + \{ (\phi \lor \psi, s) \mapsto 1 \mid (\phi, s) \in \text{dom}(\text{SEM}(\phi)_\eta) \} \]
\[ + \{ (\phi \lor \psi, s) \mapsto 2 \mid (\psi, s) \in \text{dom}(\text{SEM}(\psi)_\eta) \} \]
\[
\text{SEM}([a]\phi)_\eta = \text{SEM}(\phi)_\eta \\
+ \{([a]\phi, s) \mapsto \ast \mid (\phi, s) \in \text{dom}(\text{SEM}(\phi))_\eta\} \\
\text{SEM}(\langle a \rangle \phi)_\eta = \text{SEM}(\phi)_\eta \\
+ \{((\langle a \rangle \phi, s) \mapsto s' \mid s \xrightarrow{a} s' \land (\phi, s') \in \text{dom}(\text{SEM}(\phi))_\eta\} \\
\text{SEM}(\nu X.\phi)_\eta = \text{SEM}(\phi)_\eta[X:=\text{sem}(\phi,\eta)] \\
\text{SEM}(\mu X.\phi)_\eta = \text{ITER}_X(\phi, \eta, \{\}) \\
\text{ITER}_X(\phi, \eta, \Sigma) = \text{let} \Sigma' := \text{SEM}(\phi)_\eta[X:=\text{dom}(\Sigma)] \text{ in} \\
\text{if} \Sigma = \Sigma' \text{ then } \Sigma \text{ else } \text{ITER}_X(\phi, \eta, \Sigma')
\]
Checking strategies

Easy algorithm for checking whether a strategy is correct:

Run the strategy until you find a loop (hit the same formula at the same state again): then check whether the highest priority inside the loop is even (good) or odd (strategy is wrong!).

Can be implemented as simple recursive traversal.
Example: mutual exclusion (from Huth and Ryan 2004)

**Safety** “Only one process is in its critical section at any time.”

\[ \forall Z. \neg (c_1 \land c_2) \land [a]Z \]

**Liveness** “Whenever any process requests to enter its critical section, it will eventually be permitted to do so.”

\[ \forall Z. (\neg t_1 \lor (\mu X. c_1 \lor [a]X)) \land [a]Z \]
Mutex safety: sample run

% ./micromu.native huth-fig3.7.lts huth-fig3.7-safety.mu
Z =nu/1= ~c1 \/ ~c2 \/ ([a]Z)

### Execution time Mu.sem: 0.000000s
result: 0 1 2 3 4 5 6 7

### Execution time Strat.sem: 0.003333s
...

verifying for good state 0:
TRAV Z , 0 , loop-search
TRAV ~c1 \/ ~c2 \/ ([a]Z) , 0 , loop-search
TRAV ~c1 \/ ~c2 , 0 , loop-search
TRAV ~c1 , 0 , loop-search
TRAV ~c1 , 0 , loop-search
TRAV ~c1 , 0 , maxprio 0 here 0
TRAV ~c1 , 0 , maxprio 0 here 0
TRAV ([a]Z) , 0 , loop-search
state 0: true
...
verifying for good state 5:
TRAV Z , 5 , loop-search
TRAV ~c1 \/ ~c2 \/ ([a]Z) , 5 , loop-search
TRAV ~c1 \/ ~c2 , 5 , loop-search
TRAV ~c1 , 5 , loop-search
TRAV ~c1 , 5 , loop-search
TRAV ~c1 , 5 , maxprio 0 here 0
TRAV ~c1 , 5 , maxprio 0 here 0
TRAV ([a]Z) , 5 , loop-search
TRAV Z , 4 , loop-search
TRAV ~c1 \/ ~c2 \/ ([a]Z) , 4 , loop-search
TRAV ~c1 \/ ~c2 , 4, loop-search
TRAV ~c2 , 4, loop-search
TRAV ~c2 , 4, loop-search
TRAV ~c2 , 4, maxprio 0 here 0
TRAV ~c2 , 4, maxprio 0 here 0
TRAV ([a]Z), 4, loop-search
state 5: true
verifying for good state 6:
state 6: known good
verifying for good state 7:
state 7: known good

### Execution time verify: 0.000000s
result: 0 1 2 3 4 5 6 7
verification passed
Mutex safety: strategy run
Mutex liveness: sample run

% ./micromu.native huth-fig3.7.lts huth-fig3.7-liveness.mu
X =mu/4= c1 \ ([a]X)
Z =nu/1= \sim t1 \ X /\ ( [a]Z)

### Execution time Mu.sem: 0.000000s
result:

The formula holds nowhere: the strategy is empty. The mutex does not guarantee liveness. Why?
Mutex liveness: generating counterexamples

To generate a counterexample, we can tell micromu to negate the formula using \(-c\):

\[
% \text{/micromu.native } -c \text{ huth-fig3.7.lts huth-fig3.7-liveness.mu}
\]

### Execution time Mu.sem: 0.000000s
### Execution time Strat.sem: 0.003333s
### Execution time gendot: 0.006666s
### Execution time verify: 0.003333s
result: 0 1 2 3 4 5 6 7
verification passed

Looking at the strategy, we can find the counterexample.
Mutex liveness: counterexample strategy

The big loop corresponds to
0 → 1 → 3 → 7 → 1 → 3 → 7 → · · ·
Fixing the mutex

% ./micromu.native huth-fig3.8.lts huth-fig3.7-liveness.mu
...
result: 0 1 2 3 4 5 6 7 8
verification passed
Implementation

**compact:** about 1 kLOC OCaml (another 1 kLOC thrown away during development), no external dependencies

**quick:** “worst-case” exponential example\(^1\) \(G_1\) takes 0:02:40 and uses 28 MB RAM (down from 3+ hours / 6+ GB...)

**simple, recursive algorithms:** verifier should be easy to port to proof assistant (Coq)

---

\(^1\)15 nested quantifiers, cf. Friedmann 2009, section 5ff.
Systems of equations

First version actually substituted variables inside $\mu$-formulae: consumes exponential amount of memory with nested formulae.

Rewrite equations as a ordered system of equations:

$$\nu Z. (\neg t_1 \lor (\mu X. c_1 \lor [a]X)) \land [a]Z$$

$$Z \overset{\nu}{=} (\neg t_1 \lor X) \land [a]Z$$

$$X \overset{\mu}{=} c_1 \lor [a]X$$

Need order to restore original formula.

Vastly simplifies implementation: makes $\nu$-case trivial, $\mu$-case a lot easier.
Optimizations

- Caching of results for $\nu X.\phi$ case
- Avoiding OCaml polymorphic compare (\texttt{compare_val})
- Using maps for strategies, not association lists (requires careful strategy update)
- Caching of verified states (else easily quadratic runtime)
- Very helpful: \texttt{ocamlcp(1)/ocamlprof(1)} and \texttt{perf(1)}
Future: formal verification

• The checker is meant to be a certified decision procedure
• Formally verifying the checker to be correct results in a verified implementation of $\mu$-calculus
• Done so far: definitions of least and greatest fixpoints (on arbitrary sets), specialized version of Knaster-Tarski, $\mu$-calculus set semantics
• To do: serialize strategies into Coq terms
• To do: implement checker for strategies (using finite sets)
• To do: prove checker correct
• To do: extract verified checker?
Questions?

Thank you.
References

